Integer Minimum or Maximum Element Search Using Streaming SIMD Extensions

Version 2.1

01/99

Order Number: 243638-002

Table of Contents

Revision History

Revision	Revision History	Date
2.1	FCS revision	01/99

1 Introduction

Streaming SIMD Extensions for the Intel® Architecture (IA) provides floating point single-instruction, multiple-data (SIMD) instructions and provide additional SIMD integer instructions. These instructions provide a means to accelerate operations typical of 3D graphics, real-time physics, and spatial (3D) audio. This application note shows an example of using the additional SIMD integer instructions to traverse an array to find the minimum or maximum integer. Examples of code that exploit the Streaming SIMD Extensions are included.

The function presented in this paper finds either the minimum or maximum value in an array depending on whether `pminsw` or `pmaxsw` instructions are used in the algorithm. To avoid confusion, the rest of the document only discusses the minimum function.

This application note provides a comparison of the Min/Max algorithm implemented in C versus an implementation in assembly language using Streaming SIMD Extensions. The performance differences are noted for each implementation in Section 3.

2 The Min/Max Function

MIN and MAX functions are used to find the value and location of the smallest or largest value in an array. This applications note focuses on an implementation of finding the smallest short integer (16 bit) value in an array (the MIN function). The function returns the array index of the value.

2.1 Applications for a Min/Max Function

There are many applications that need to search for the largest or smallest number in an array. Voice recognition is one such application.

2.2 Implementing the MIN Function

The MIN function finds the minimum integer in the array and saves the index of this value. If there are several numbers in the array that match this value, the index returned is the lowest number.

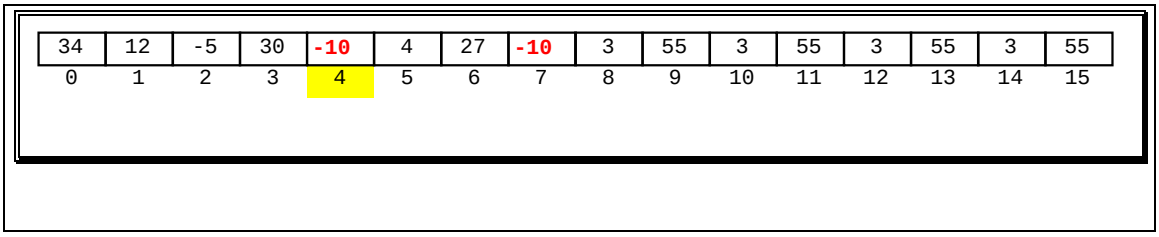

Figure 1: Example array showing two occurrences of a minimum value

For example, in Figure 1, the value -10 occurs twice (at indexes 4 and 7); the index returned by MIN is 4.

In C, a MIN function is rather straightforward. Example 1 shows a fragment of C code for calculating the minimum integer.

```
WORD tmpMin = a[0];

   indexOfMin = 0;

   for (int i=1; i<nVals; i++) {
      if (a[i] < tmpMin) {
         indexOfMin = i;
         tmpMin = a[i];
      }
   }
```

Example 1: C code for finding the minimum value of an array

Two variables keep track of the current minimum (tmpMin) and its index (indexOfMin). The code initializes the index variable to 0 and the current minimum to the first element value. Then for each element in the array, it compares the value with the current minimum value. If the element value is less than the current minimum, the current minimum is replaced with the element value and the index with the element index. Note that if the element matches the current minimum value, no values are replaced. This has the desired results of only capturing the first occurrence of a minimum value.

Using SIMD instructions allows processing four values in parallel. In the loop shown in Example 2, the mm7 register is the equivalent of tmpMin and the QWORD pointed to by [edi] is the equivalent of indexOfMin. The mm2 register contains the indices corresponding to the values located in mm0. The mm4 register contains four copies of the number 4, which is used to update mm2. Note that an explicit reference to [edi] does not exist in the code; the maskmovq instruction uses [edi] as an implicit destination pointer.

```
loop_top:
       movq mm0, -8[edx+ecx*8]
       dec    ecx

       pminsw    mm7, mm0  // get mins
       pcmpeqw   mm0, mm7  // create mask
       maskmovq  mm2, mm0  // store indicies of mins
       psubw     mm2, mm4

       jnz    loop_top
```

Example 2: Streaming SIMD Extensions loop for finding the minimum value

The first instruction in the loop, movq, loads array data into mm0.

The mm7 register contains the most recent minimum values found from the array. Figure 2 shows how a single pminsw instruction calculates the minimum integers of two MMX registers. The rest of the loop deals with keeping track of the indices of these four minimum values.

PMINSW MM7, MM0

-10	27	4	-10	mm0
55	3	55	3	mm7
-10	3	4	-10	mm7

Note: shaded items are the minimum values

Figure 2: PMINSW instruction for calculating four minimum values simultaneously

Figure 3 shows that the new minimums (`mm7`) are compared with `mm0` to determine contributing values. Any words in `mm0` that match `mm7` are replaced with all 1's. Zeroes in `mm0` mean that the corresponding current minimums in `mm7` came from a previous iteration of the loop.

PCMPEQW MM0, MM7

-10	3	4	-10	mm7
-10	27	4	-10	mm0
FFFF	0	FFFF	FFFF	mm0

Figure 3: Create mask corresponding to new values

The `maskmovq` instruction updates the indices based on any contributing values from `mm0`.

MASKMOVQ MM2, MM0

FFFF	0	FFFF	FFFF	mm0
7	6	5	4	mm2
7	10	5	4	[edi]

Figure 4: Replace index values

By starting at the highest index and working toward the lowest index the desired result is achieved. The remainder of the code consolidates the four minimum values in `mm7` and their corresponding indices in [edi]. The minimum of those four values that has the lowest index is the result of the function.

2.2.1 Tips and Tricks

Using the new `maskmovq` instruction helps spread out the algorithm's computations to the various execution units of the processor. The `maskmovq` instruction uses the memory ports to perform the equivalent of an `"and -> andnot -> or"` operation to correctly modify the SIMD indexOfMin field.

3 Performance

To find the minimum or maximum integer in an array requires the traversal of an array. C code naturally accesses each integer one at a time. The additional SIMD integer instructions (a part of the Streaming SIMD Extensions) can access four integers at a time.

3.1 Gains/Improvements

The benefits of Streaming SIMD Extensions are:

- No branches in the loop except for the loop control
- Four words of data are processed per loop
- Reduced number of instructions compared to MMX™ technology
- Prefetch instructions to reduce memory latency effect

The assembly code shown in Example 3 was generated by a C compiler. Note the conditional jump after the first `cmp` instruction.

```
loop_top:
        cmp       cx, WORD PTR a[eax*2]
        jle       skip_replace

        mov       WORD PTR indexOfMin, ax
        mov       cx, WORD PTR a[eax*2]
skip_replace:
        inc       eax
        cmp       eax, 4096
        jl        loop_top
```

Example 3: Loop generated by a C compiler (annotated)

The Streaming SIMD Extensions version of the loop eliminates this instruction. The `pminsw` instruction takes the place of several MMX™ instructions. Example 4 shows the MMX™ instructions needed to duplicate the capability of the `pminsw` instruction.

```
movq mm3,  mm7        ; Get the minimum of mm7 and mm0 into mm7
pcmpgtw mm7, mm0      ; minimum contd.
pxor mm0, mm3         ; minimum contd.
pand mm7, mm0         ; minimum contd.
pxor mm7, mm3         ; minimum done
```

Example 4: MMX™ instructions to calculate the minimum

3.2 Considerations

Memory alignment and array size are two things to consider when deciding whether to implement this algorithm. A small array would not benefit much because of the setup and cleanup code. Unaligned data would run into problems with this code as written due to the penalty incurred for cache-line splits (accessing data from two cache lines with one instruction). To use this algorithm on unaligned data,

additional code is needed in the setup and cleanup portions of the program to cause the loop to operate on aligned data.

Prefetching instructions should be used to make sure data is in the data cache. (Include this in the study.) Position the prefetches in the code and measure the performance in the C code and the code using Streaming SIMD Extensions. Include these results in the summary table.

4 Conclusion

The use of floating point data is not the only thing to look at when deciding to optimize code with the Streaming SIMD Extensions. This instruction set includes several new SIMD integer instructions. The Kamtai New Instructions, `pminsw` and `pmaxsw`, are two of these instructions that can increase the performance of integer code.

5 C Coding Example

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <xmmintrin.h>

typedef short WORD;     // array element data type

union M64 {             // define union of 4 WORDs and 1  __m64
  WORD w[4];
  __m64 m1;
};

const WORD nVals = 0x1000;// array size

// align data to avoid splits across two cache lines
__declspec(align(32)) WORD a[nVals];                  // the array
M64 mins;               // holds 4 minimum values
M64 idxs;               // holds the indices of the mins
M64  maxes    = {0x7fff, 0x7fff, 0x7fff, 0x7fff}; // maximum integer values

WORD indexOfMin;        // the lowest index of minimum value


const WORD fours[4] =  { (WORD)4, (WORD)4, (WORD)4, (WORD)4 };

/***************** The C version ******************/
void cmin() {
  WORD tmpMin = a[0];   // holds the current min, start with a[0]

  indexOfMin = 0;       // and its index

  for (int i=1; i<nVals; i++) {  // increment through array
    if (a[i] < tmpMin) {         // new min?
      indexOfMin = i;            // replace index and value
      tmpMin = a[i];
    }
  }
}
/*************************************************/
```

6 Streaming SIMD Extensions Assembly Code Example

```
/********* The Streaming SIMD Extensions version ******/
void mmxmin() {
  idxs.w[3] = nVals-1;    // Initialize indices with last 4 of array
  idxs.w[2] = nVals-2;
  idxs.w[1] = nVals-3;
  idxs.w[0] = nVals-4;
    __asm {
      push edi
      movq mm4, fours     // 4,4,4,4 in mm4
      lea    edi, idxs    // point edi to idxs (for maskmovq)
      movzx  ecx, WORD PTR nVals // number of WORDs
      sar    ecx, 2       // number of QWORDs
//    prefetcht0  -72[edx+ecx*8] // bring five iteration ahead
      lea    edx, a       // edx points to array
      movq mm7, -8[edx+ecx*8] // start with end of array
      movq mm2, [edi]     // indices in mm2
      movq mm5, [edi]     // indices in mm2
      dec  ecx

movq   mm0, -8[edx+ecx*8]
      dec       ecx
      psubw     mm2, mm4
      pminsw    mm7, mm0  // get mins
      pcmpeqw   mm0, mm7  // create mask
      maskmovq  mm2, mm0  // store indicies of mins
      paddw  mm4, mm4     // constant now contains four “8”

loop_top:
/* Two iteration of the loop are done in parallel, the effect of almost
  a two time speed up achieved by using extra registers and pairing up
  instruction to achive a minimal waist on a 16 bit instruction stream
  that is being fed to the decoders
*/
      movq   mm0, -8[edx+ecx*8]  // load next four numbers
      movq   mm1, -16[edx+ecx*8]// same as above
      psubw     mm2, mm4    // subtract 8 from each index to get next index
      psubw     mm5, mm4    // same as above
      pminsw    mm7, mm0    // get mins
      pminsw    mm7, mm1
      pcmpeqw   mm0, mm7    // create mask
      pcmpeqw   mm1, mm7    // create mask
//    prefetcht0  -96[edx+ecx*8] // bring 6 iteration ahead
      maskmovq  mm5, mm0    // store indicies of mins
      maskmovq  mm2, mm1
      sub       ecx, 0x2    // decrement counter by 2                jnz
    loop_top    // is ecx non-zero

                      // clean up

      movq mm2, [edi]     // get updated indices
```

```
        pshufw mm0, mm7, 0xe  // min upper 2 with lower 2
        pminsw mm0, mm7

        pshufw mm1, mm0, 1    // min upper 1 with lower 1
        pminsw mm1, mm0

        pshufw mm1, mm1, 0    // broadcast min

        movq mins, mm1   // save the min value

        pcmpeqwmm7, mm1    // create mask
        pand mm2, mm7    // indices of corresponding mins
        pandn  mm7, maxes     // load the rest with "maxint"
        por    mm7, mm2    // merge

        pshufw mm0, mm7, 0xe  // min upper 2 with lower 2
        pminsw mm0, mm7

        pshufw mm1, mm0, 1    // min upper 1 with lower 1
        pminsw mm1, mm0

        pshufw mm1, mm1, 0    // broadcast index
        movq [edi], mm1         // idxs of min values
        mov    ax, [edi]
        mov    indexOfMin, ax   // save result

        emms
        pop    edi
    }
}
/************************************************/
```

6.1 Streaming SIMD Extensions Assembly with MMX Technology for Inner Loop

```
void mmxmin() {
   idxs.w[3] = nVals-1;
   idxs.w[2] = nVals-2;
   idxs.w[1] = nVals-3;
   idxs.w[0] = nVals-4;
     __asm {
        push edi
        movq mm4, fours
        lea    edi, idxs
        movzx  ecx, WORD PTR nVals
        sar    ecx, 2
        lea    edx, a
        movq mm7, -8[edx+ecx*8]
        movq mm2, [edi]     // indices
        movq mm6, mm2    // indices of current mins              psubw mm2,
mm4     // adjust them down
        dec    ecx
        movq mm0, -8[edx+ecx*8]
```

```
loop_top:                   // MMX instructions only
                            // in inner loop
              dec    ecx

        movq mm3, mm0    // save new values
        pcmpgtwmm0, mm7    // 1's keep old mins
        pxor mm7, mm3    // xor data together
        pand mm7, mm0    // keep old mins
        pxor mm7, mm3    // merge
        pand mm6, mm0    // keep old indexes
        pandn  mm0, mm2    // mask new indexes
        por    mm6, mm0    // merge
        movq mm0, -8[edx+ecx*8]
        psubw  mm2, mm4

        jnz    loop_top

                     // clean up

        movq mm2, [edi]    // get updated indices

        pshufw mm0, mm7, 0xe  // min upper 2 with lower 2
        pminsw mm0, mm7

        pshufw mm1, mm0, 1   // min upper 1 with lower 1
        pminsw mm1, mm0

        pshufw mm1, mm1, 0   // scatter

        movq mins, mm1   // save the min value

        pcmpeqwmm7, mm1    // create mask
        pand mm2, mm7    // indices of corresponding mins
        pandn  mm7, maxes    // load the rest with "maxint"
        por    mm7, mm2    // merge

        pshufw mm0, mm7, 0xe  // min upper 2 with lower 2
        pminsw mm0, mm7

        pshufw mm1, mm0, 1   // min upper 1 with lower 1
        pminsw mm1, mm0

        pshufw mm1, mm1, 0   // scatter
        movq [edi], mm1    // idxs of min values
        mov    ax, [edi]
        mov    indexOfMin, ax // index of min fixup

        emms
        pop    edi
    }
}
```

7 Coding Example Using Compiler Intrinsics for the Streaming SIMD Extensions

```
/**************** Intrinsics Version ***************/
void intrmin() {
   __m64* ptr = (__m64*)&a[nVals-4];   // array pointer
   __m64 curMin  = *ptr;               // current min
   M64  fours_init = {4,4,4,4};        // constant 4s for index update
   M64  indices  = {nVals-4, nVals-3, nVals-2, nVals-1}; // last 4
   idxs = indices;                     // working copy of indices
   __m64 fours = fours_init.m1;        // __m64 version of 4,4,4,4
   char* const edi_addr = (char*)&idxs; // for maskmovq

   for(int i=nVals-4; i>0; i-=4) {     // from array end to beginning
      __m64  nextVals  = *ptr--;       // get next QWORD
           curMin    = _m_pminsw(curMin, nextVals);    // get mins
      __m64  mask    = _m_pcmpeqw(nextVals, curMin);   //create mask
           _m_maskmovq(indices.m1, mask, edi_addr);   // store indices
           indices.m1    = _m_psubw(indices.m1,fours);// adjust indices
   }
/***** cleanup *****/
   __m64  shuf = _m_pshufw(curMin, 0xe);  // min upper 2 with lower 2
        shuf = _m_pminsw(shuf, curMin);

   __m64  shuf2  = _m_pshufw(shuf, 0x01);// min upper 1 with lower 1
        shuf2  = _m_pminsw(shuf2, shuf);

        mins.m1= _m_pshufw(shuf2, 0);    // broadcast min

   __m64  minMask = _m_pcmpeqw(curMin, mins.m1);// create mask
   __m64  minIdx = _m_por(_m_pand(minMask, idxs.m1), _m_pandn(minMask,
maxes.m1)); // replace all non-minimum indices with maximum int

        shuf = _m_pshufw(minIdx, 0xe);    // min upper 2 with lower 2
        shuf = _m_pminsw(shuf, minIdx);

        shuf2  = _m_pshufw(shuf, 0x01);  // min upper 1 with lower 1
        shuf2  = _m_pminsw(shuf2, shuf);

        idxs.m1= _m_pshufw(shuf2, 0);    // broadcast index
        indexOfMin = idxs.w[0];          // save result
}


/******* test driver *******/

int main(int argc, char **argv) {

   srand(20);    // fixed seed

   cout << "Program " << argv[0] << " starting\n";
   for (int i=0; i<nVals; i++) {
     a[i] = rand();
```

```
   }

// cmin();
// mmxmin();
   intrmin();

   cout << "Min value is at index " << indexOfMin << " Value is " <<
a[indexOfMin] << "\n";
   return 0;
}
```